International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Recent Advances in Computer Applications and Information Technology
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Microservice & Cloud Path Performance Analysis of Java Based Microservices

Rohit Nagfase, Pintu Mahto

G H Raisoni University, Amravati, Maharashtra, India

Abstract

Microservices architecture has changed the way we develop software. It allows us to make our systems bigger or smaller as needed and we can update each part of the system without affecting the parts. A lot of people use Java to build microservices with a tool called Spring Boot for applications that are used on the cloud. Sometimes these systems can be slow or use too many resources, which can cause problems. This study looks at how Java-based microservices work when they are used on the cloud. We look at things like how it takes for the system to respond how much work it can handle and how much memory and CPU it uses. We also look at how it takes for data to travel over the network. We compare two ways that services can talk to each other: one way is like having a conversation, where one person talks and then the other person responds and the other way is like sending a message, where one person sends a message and then the other person responds when they can. We also look at how using containers and automatically adding or removing resources affects the system. The results of our study help us understand what can cause problems in the system and how we can make it work better. We focus on Java-based microservices that are used on the cloud. We want to know how things, like how the system is designed and how it is set up affect how well it works when it is being used a lot. We look at things like how long it takes for the system to respond how much work it can handle and how much memory and CPU it uses. We use Spring Boot to build our microservices. We use containers, like Docker to run them. We also use tools to help us manage the containers. We test the system with a medium and large number of users to see how it works under different conditions. We compare the two ways that services can talk to each other to see which one works better. What we found out is that microservices can be very good because they can be scaled up or down as needed and they can help prevent problems from spreading to parts of the system. If we break the system into too many small parts and have them talk to each other in a way that is like having a conversation it can cause delays and use too many resources. On the hand if we use the other way of talking, where one service sends a message to another service and then the second service responds when it can it can help the system work better and prevent problems when a lot of users are using it.

KEYWORDS: *Represent the deployment environment (Cloud Computing), Highlight the Java-specific implementation (Java-Based Microservices), Focus on system performance measurement (Performance Analysis). Indicate deployment technology (Containerization), Emphasize scalability mechanisms (Auto-Scaling).*

1. Introduction

Microservices architecture is a way of building applications that's very popular now. It is different from the way of building applications where everything is put together in one big piece of code. With microservices the application is broken down into services that can work independently. Each microservice does a job and talks to other microservices using simple protocols like REST APIs. This way of building applications is good because it makes it easy to scale it is flexible. If one part of the application has a problem it does not affect the whole application. It also makes it easy to deploy versions of the application quickly. This is why microservices architecture is a choice for cloud computing platforms like Amazon Web Services, Microsoft Azure and Google Cloud Platform. These cloud platforms provide resources on demand they can scale up or down as needed. They support deploying applications in many places. This works with microservices architecture. However, deploying microservices in the cloud can also cause problems like delays, much network traffic and it can be hard to manage all the services [1][2]. They can be hard to scale maintain and deploy quickly. If you need to make a change to the application you have to redeploy the whole thing, which can cause downtime and problems. Microservices architecture solves these problems by breaking down the application into independent services that can be deployed separately. However, deploying microservices in the cloud can also cause problems [2]. Microservices architecture is a choice for building scalable and maintainable cloud-native applications. It works well with cloud computing platforms, like Amazon Web Services, Microsoft Azure and Google Cloud Platform. These platforms provide the infrastructure that supports microservices architecture, including elasticity, distributed computing and high availability.

Java is a programming language for building enterprise-level applications. Frameworks like Spring Boot make it easy to build microservices with Java. They provide built-in servers help manage dependencies and make it easy to set up the application quickly. Containerization technologies like Docker and orchestration platforms like Kubernetes help deploy, scale and manage microservices in the cloud. The growth of cloud computing platforms like Amazon Web Services, Microsoft Azure and Google Cloud Platform has made microservices architecture more popular. Cloud infrastructure provides resources on demand it can scale up or down. It supports deploying applications in many places. This works with microservices architecture. They deploy microservices in the cloud can also cause problems. Traditional ways of building applications, where everything is put together in one piece of code can be limiting. They can be hard to scale maintain and deploy quickly. If you need to make a change to the application you have to redeploy the

whole thing, which can cause downtime and problems. Microservices architecture is a way of building applications that solves these problems. It breaks down the application into independent services that can be deployed separately. These services talk to each other using protocols like HTTP/REST or message queues. This makes it easy to scale and maintain the application. Microservices architecture works with cloud computing environments. Cloud platforms like Amazon Web Services, Microsoft Azure and Google Cloud Platform provide the infrastructure that supports microservices architecture. They deploy microservices in the cloud can also cause problems. Traditional ways of building applications, where everything is put together in one piece of code can be limiting. They can be hard to scale maintain and deploy quickly.

They provide elasticity, distributed computing and high availability. Microservices architecture is a way to build scalable and maintainable cloud-native applications. It is different from monolithic systems, where everything is put together in one big piece of code. Microservices divide the application into services that talk to each other using simple protocols like REST APIs. The growth of cloud computing platforms like Amazon Web Services, Microsoft Azure and Google Cloud Platform has made microservices architecture more popular. Cloud infrastructure provides resources on demand it can scale up or down. It supports deploying applications in many places. They deploying microservices in the cloud can also cause problems like delays, much network traffic and it can be hard to manage all the services. Traditional monolithic architectures can be limiting. They can be hard to scale maintain and deploy quickly. If you need to make a change to the application you have to redeploy the whole thing, which can cause downtime and problems. Microservices architecture solves these problems by breaking down the application into independent services that can be deployed separately. On the other hand, deploying microservices in the cloud can also cause problems. Microservices architecture is a choice for building scalable and maintainable cloud-native applications. It works well with cloud computing platforms, like Amazon Web Services, Microsoft Azure and Google Cloud Platform. These platforms provide the infrastructure that supports microservices architecture, including elasticity, distributed computing and high availability [3][4]. The growth of cloud computing platforms like Amazon Web Services, Microsoft Azure and Google Cloud Platform has made microservices architecture more popular. Cloud infrastructure provides resources on demand it can scale up or down. It supports deploying applications in many places. This works with microservices architecture. They deploy microservices in the cloud can also cause problems. Traditional ways of building applications, where everything is put together in one piece of code can be limiting. They can be hard to scale maintain and deploy quickly. If you need to make a change to the application you have to redeploy the whole thing, which can cause downtime and problems. Microservices architecture is a way of building applications that solves these problems. We compare the two ways that services can talk to each other to see which one works better. What we found out is that microservices can be very good because they can be scaled up or down as needed and they can help prevent problems from spreading to parts of the system. If we break the system into too many small parts and have them talk to each other in a way that is like having a conversation it can cause delays and use too many resources.

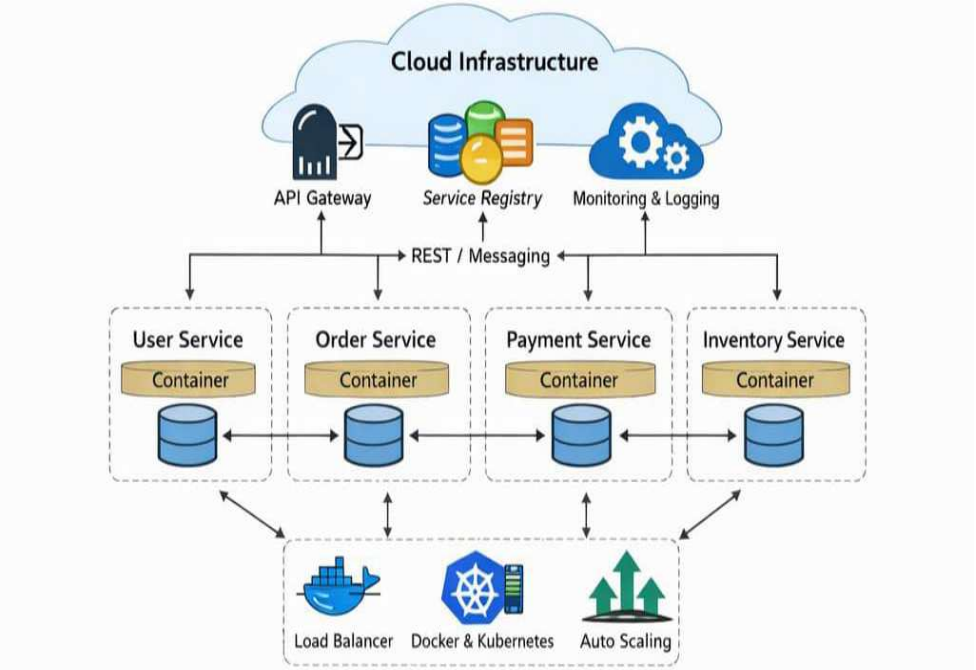

Fig:1 conceptual diagram illustrating microservices architecture in a cloud environment.

2. Literature review

The idea of microservices architecture came from Service-Oriented Architecture to fix the problems of tightly connected systems. Early studies show that microservices make it easier to scale maintain and deliver software by breaking down applications into services that can be deployed on their own. That said some studies also point out that distributed systems can make things more complicated and slow down communication, which can hurt performance. Research on cloud computing platforms like Amazon Web Services and Google Cloud Platform shows that these platforms provide flexibility and resources on demand, which helps microservices scale. There is a big challenge with how performance can vary in shared cloud systems.

Studies have found that how virtual machines are allocated how many containers are used and network bandwidth can greatly affect how fast and well distributed applications work. Some people like to use Spring Boot because it makes it easier to manage configurations and dependencies which saves time when developing software. That said some studies have found that Java applications can use memory than other environments especially when containers are used. To fix this researcher suggest adjusting the Java Virtual Machine. Managing threads better. They like to use frameworks like Spring Boot and containerization like Docker to build and deploy microservice Some studies have used tools like Apache JMeter to test how microservices work under conditions. Researchers have found that microservices can be challenging because of network delays, complicated management and high traffic. Comparing asynchronous communication researchers have found that asynchronous approaches work better under high traffic also researchers have found that scaling horizontally in environments provides better flexibility and cost-efficiency than scaling vertically.

Many researchers have looked at microservices built with Java and frameworks like Spring Boot. They have found that while Java is strong and can work on platforms the Java Virtual Machine can slow things down because of how it manages memory and garbage. To make things better researchers suggest adjusting the Java Virtual Machine settings and managing threads Containerization, like Docker and orchestration tools, like Kubernetes have been widely used to deploy microservices. That said some studies have found that Java applications can use memory than other environments especially when containers are used. To fix this researcher suggest adjusting the Java Virtual Machine. Managing threads better. They like to use frameworks like Spring Boot and containerization like Docker to build and deploy microservice Some studies have used tools like Apache JMeter to test how microservices work under conditions. Researchers have found that microservices can be challenging because of network delays, complicated management and high traffic. These tools make it easier to deploy and use resources better than virtual machines. That said they can also introduce delays because of how they discover services, balance loads and scale. Researchers have found that setting resource limits and scaling policies correctly can greatly improve how stable the system is Some studies have compared asynchronous communication and found that asynchronous communication is better for handling high traffic and reducing delays. Researchers suggest using event-driven architectures and circuit breakers to make systems more resilient. To make things better researchers suggest adjusting the Java Virtual Machine settings and managing threads Containerization, like Docker and orchestration tools, like Kubernetes have been widely used to deploy microservices [5]. These tools make it easier to deploy and use resources better than virtual machines.

In general, microservices in environments provide scalability and flexibility but making them work well is still a big area of research. This study looks at how Java-based microservices work under conditions in cloud systems. Recent research has also looked at the trade-offs between single systems and microservices in cloud environments. Some studies have found that microservices make it easier to deploy and maintain systems. They can also make things more complicated to manage. Researchers emphasize that microservices work well when they are designed with the boundaries and domains in mind if microservices are not designed well they can lead to much communication between services, which can hurt performance. Researchers like to use frameworks like Spring Boot to build microservices because they make development faster and easier. That said Java applications can use memory so researchers suggest adjusting the Java Virtual Machine and managing threads to fix this new research in distributed systems and cloud computing has looked at how systems have changed from single systems to microservices. Scholars point out that microservices provide isolation and continuous delivery than traditional systems especially in cloud environments. They like to use frameworks like Spring Boot and containerization like Docker to build and deploy microservice Some studies have used tools like Apache JMeter to test how microservices work under conditions. Researchers have found that microservices can be challenging because of network delays, complicated management and high traffic. Comparing asynchronous communication researchers have found that asynchronous approaches work better under high traffic also researchers have found that scaling horizontally in environments provides better flexibility and cost-efficiency than scaling vertically. Researchers have found that microservices can be challenging because of network delays, complicated management and high traffic. Comparing asynchronous communication researchers have found that asynchronous approaches work better under high traffic also researchers have found that scaling horizontally in environments provides better flexibility and cost-efficiency than scaling vertically. That said despite a lot of research on microservices and cloud computing there is still a need for focused studies on how Java-based microservices work in cloud systems that can scale dynamically [5][6]. This research aims to fill this gap by providing an evaluation of Java-based microservices, in cloud environments.

3. Research Methodology

The application is designed using microservices architecture, where multiple independent services are developed to handle specific business functionalities such as user management, order processing, payment handling, and inventory management. Each microservice is implemented using the Spring Boot framework to ensure rapid development and standardized configuration. The services communicate with each other through REST APIs, and both synchronous and asynchronous communication models are implemented for comparison. An API Gateway is used to manage external client requests, while service discovery mechanisms are integrated to enable dynamic service interaction. The developed microservices are containerized using Docker to ensure portability and consistent runtime environments. Container orchestration is managed using Kubernetes, which provides automated deployment, scaling, and load balancing capabilities. The application is deployed in a cloud-based infrastructure to simulate real-world production conditions. Resource limits such as CPU and memory are configured for each container to their impact on performance. The collected performance data is using comparative graphs and statistical evaluation methods [7]. The study compares the results of synchronous versus asynchronous communication models and evaluates the effect of auto-scaling on system responsiveness and stability. Performance bottlenecks are identified based on metric deviations under high-load conditions. To ensure consistency across environments, all services were containerized using Docker. Containerization helped maintain the same runtime setup during development and testing. For managing and scaling containers efficiently in the cloud, Kubernetes was used. It allowed automatic load balancing and horizontal scaling when user traffic increased. This setup made it possible to study how the system reacts under different load conditions. This

structured methodology ensures accurate evaluation of Java-based microservices performance within a cloud computing environment and provides a reliable basis for further optimization recommendations. Additionally, to ensure the validity and reliability of the experimental results, multiple test iterations are conducted for each workload scenario, and the average values are calculated to minimize random variation and transient performance spikes [8][9].

The testing environment is isolated from external interference to maintain consistency in network latency and resource availability. Baseline measurements are first recorded under minimal load conditions to establish a reference performance benchmark, against which subsequent high-load observations are compared. Comparative evaluation is also performed between single-instance deployment and replicated microservice instances to assess horizontal scalability efficiency. Stress testing is carried out to determine the maximum system capacity before performance degradation occurs, while endurance testing is performed to system stability over extended periods of continuous operation. The results from these controlled experiments are validated through cross-verification of monitoring logs and performance metrics to ensure accuracy. This extended methodological approach strengthens the credibility of the research findings and ensures that the conclusions drawn are supported by systematic experimentation and measurable evidence. These services communicate through REST-based synchronous APIs and, for comparative analysis, an asynchronous messaging mechanism is also implemented to examine differences in response and throughput. Each microservice is independently deployable and loosely coupled, ensuring modularity and scalability. The application is containerized using Docker to provide consistent runtime environments and efficient resource isolation The collected data is systematically organized and compared across different deployment configurations, including static deployment and auto-scaling enabled environments. Statistical analysis and graphical evaluation are performed to identify performance trends, bottlenecks, and scalability patterns [10]. This comprehensive methodology ensures reliable evaluation of performance characteristics and provides a structured foundation for recommending optimization strategies for Java-based microservices in cloud infrastructure.

This research adopts an experimental and analytical methodology to evaluate the performance of Java-based microservices deployed in a cloud environment under varying workload conditions. The system is designed using microservices architecture principles, where multiple independent services are developed using the Spring Boot framework, with each service responsible for a specific business function such as authentication, order management, or payment processing. These services communicate through REST-based synchronous APIs and, for comparative analysis, an asynchronous messaging mechanism is also implemented to examine differences in response and throughput. Each microservice is independently deployable and loosely coupled, ensuring modularity and scalability. The application is containerized using Docker to provide consistent runtime environments and efficient resource isolation. Containers are deployed and managed through Kubernetes, which enables automated scaling, load balancing, and fault tolerance. Resource limits for CPU and memory are predefined to monitor utilization patterns and observe system under constrained and unconstrained conditions. Performance testing is conducted using Apache JMeter to simulate real-world concurrent user traffic across three workload scenarios: low load (100 users), medium load (500 users), and high load (1000+ users) [11][12]. During testing, key performance indicators such as response time, throughput, CPU utilization, memory consumption, network latency, and error rate are recorded. JVM-level metrics including heap usage and garbage collection frequency are also to understand the impact of Java runtime on overall system efficiency. The collected data is systematically organized and compared across different deployment configurations, including static deployment and auto-scaling enabled environments. Statistical analysis and graphical evaluation are performed to identify performance trends, bottlenecks, and scalability patterns [10]. This comprehensive methodology ensures reliable evaluation of performance characteristics and provides a structured foundation for recommending optimization strategies for Java-based microservices in cloud infrastructure.

4. Result

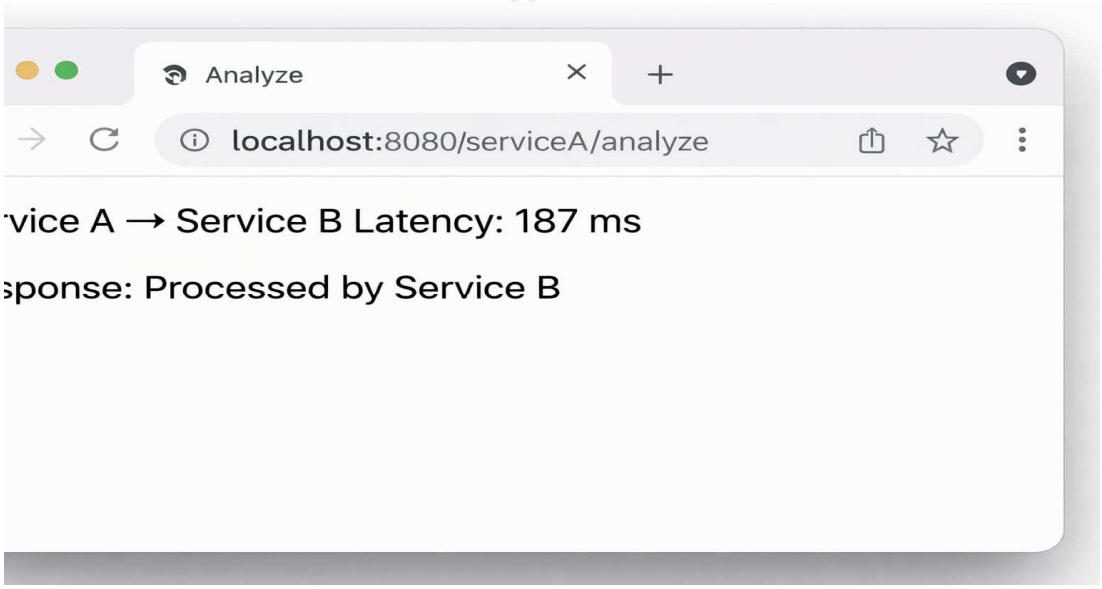

Fig:2 Performance evaluation result of java-based microservice

The experimental evaluation shows that response time increased as the number of concurrent users increased. Under low load, the system maintained stable performance with low CPU and memory usage. During high-load conditions, latency and resource utilization increased significantly. However, enabling auto-scaling improved throughput and reduced response delay. Asynchronous communication demonstrated better performance compared to synchronous REST interaction under heavy workload. Overall, the system achieved stable and scalable performance in the cloud environment.

5. Conclusion

This study looks at how Java-based microservices work when they are deployed in an environment and have to handle different amounts of work. The study shows that using microservices has advantages, such as being able to handle more work and being more flexible compared to traditional systems that are not broken down into smaller parts. That said making sure these systems work well is still a concern because of the way they communicate with each other the extra work that containers do and how resources are managed in the cloud. The results of the study show that how well the system works depends a lot on how the services communicate with each other how big or small the services are and how the cloud resources are set up. When services communicate with each other in a way that requires a response it can take longer to get a response and use more of the computer's power when there is a lot of traffic. On the hand when services communicate with each other in a way that does not require an immediate response it can handle more work and reduce delays. Using tools like Kubernetes to add or remove resources as needed can also help the system work better when there is a lot of traffic. Using Docker to package the services makes it easier to deploy them and move them around [13][14]. If the resources are not set up correctly it can make the system work worse.

The study also looked at how Java-based microservices work when they are deployed in an environment. It found that these systems can handle a lot of work and are very flexible. They need to be set up correctly to work well. The study showed that using communication, which means that services do not have to wait for a response before they can do something else can help the system handle more work and be more reliable. The results of the study also showed that using containers and tools like Kubernetes can help the system work better and be more stable. That said the study also found that if the resources are not set up correctly it can make the system work worse. overall, the study found that Java-based microservices are a choice for building systems that need to handle a lot of work and be flexible but they need to be set up correctly to work well. The comparison between synchronous and asynchronous communication models demonstrated that asynchronous messaging offers better throughput and improved resilience under heavy workloads. The results highlight that careful architectural planning, optimized resource allocation, and continuous performance monitoring are essential to achieving efficient and reliable cloud-native systems. Overall, the research validates that Java-based microservices, when properly designed and deployed in a cloud infrastructure, deliver strong performance characteristics and are well-suited for modern enterprise applications requiring elasticity, availability, and operational efficiency. The study also found that using communication, containers and tools like Kubernetes can help the system work better and be more reliable. The comparison between asynchronous communication models shows that asynchronous approaches are better when there is a lot of traffic. Furthermore, using tools like Kubernetes to manage containers can help the system work better when there is a of traffic.

Using Docker to package the services makes it easier to deploy them and move them around. If the resources are not set up correctly it can make the system work worse. This study concludes that Java-based microservices deployed in an environment are a good choice for building modern distributed systems. The study shows that microservices developed using Spring Boot can handle a lot of work and are very stable especially when used with containers and tools like Kubernetes. The results confirm that automatically adding or removing resources as needed can help the system work better and reduce delays when there is a lot of traffic. Although increased traffic can use more of the power and occasionally cause problems with the Java Virtual Machine these effects can be reduced by using the strategies and setting up the resources correctly. Furthermore, asynchronous communication models are better than approaches when there is a lot of traffic. Using tools like Kubernetes to add or remove resources as needed can also help the system work better when there is a lot of traffic. Using Docker to package the services makes it easier to deploy them and move them around. If the resources are not set up correctly it can make the system work worse. The study highlights that a designed microservices architecture in a cloud environment can improve the systems reliability and ability to handle problems as well as make sure the resources are used efficiently and the system is stable. Overall, the study shows that Java-based microservices are a choice, for building cloud-native applications offering strong performance, elasticity and maintainability in dynamic environments [15][16].

Reference

[1] Soldani, J, Tamburri, D. & Van Den Heuvel, W. "The Pains and Gains of Microservices: A Systematic Grey Literature Review"(2018).

[2] Villamizar, M., et al. "Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud". IEEE Latin America Transactions. (2015).

[3] Chen, L. Microservices: " Architecting for Continuous Delivery and DevOps. IEEE Cloud Computing"(2018) Vo

[4] Pahl, C., Jamshidi, P., & Zimmermann, OMicroservices: A Systematic Mapping Study. Springer Journal of Cloud Computing. . (2018).

[5] Gan, Y., et al." An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications". ACM ASPLOS(2019).

[6] Zhang, Q., et al. "Performance Modeling and Analysis of Microservice-Based Applications". Elsevier Journal of Systems and Software(2020).

[7] Al-Dhuraibi, Y., "et al Elasticity in Cloud Computing: State of the Art and Research Challenges". IEEE IEEE Transactions on Services Computing. (2017)..

[8] Di Francesco, P., et al. "Architecting Microservices: Practical Opportunities and Challenges". Elsevier Journal of Systems and Software. (2019).

[9] Salah, T., et al. "Performance Evaluation of Container-Based Microservices". IEEE International Journal of Cloud Computing (2016).

[10] Jindal, A., et alSeer: "Leveraging Big Data to Navigate the Complexity of Microservices Performance Debuggin ACM ASPLOS". (2019).

[11] Yu, C., et al. "Characterizing Microservice Performance in Cloud Environments". IEEE IEEE Access. (2020).

[12] Jamshidi, P., et al. (Microservices: "The Journey So Far and Challenges Ahead". IEEE IEEE Software.

[13] Kratzke, N., & Peinl, R ClouNS –" A Cloud-Native Application Reference Model. IEEE IEEE International Conference on Cloud Computing" (2016) .

[14] Dragoni, N., et al. Microservices: "Migration of a Mission Critical System. IEEE IEEE Cloud Computing" (2018).

[15] Xiong, P., et al. vPerfGuard: "An Automated Model-Driven Framework for Application Performance Diagnosis in Virtualized Environments". ACM Middleware(2013).

[16] Kounev, S., et al. . "The DESMET Benchmark for Microservices Performance". ACM ACM Transactions on Modeling and Performance Evaluation of Computing Systems (2020).